GTSuite Report Server

Appendix - FastReport PDF Generation Service

Standalone Report Generation Engine

GTsoftware di Giancarlo Thiella

Generated: November 20, 2025

Overview

The GTSuite Report Server is a standalone Delphi application that provides professional PDF report generation using FastReport VCL. It operates as an independent HTTP service that receives report templates and data from the Node.js server, generates high-quality PDFs, and returns them for client download.

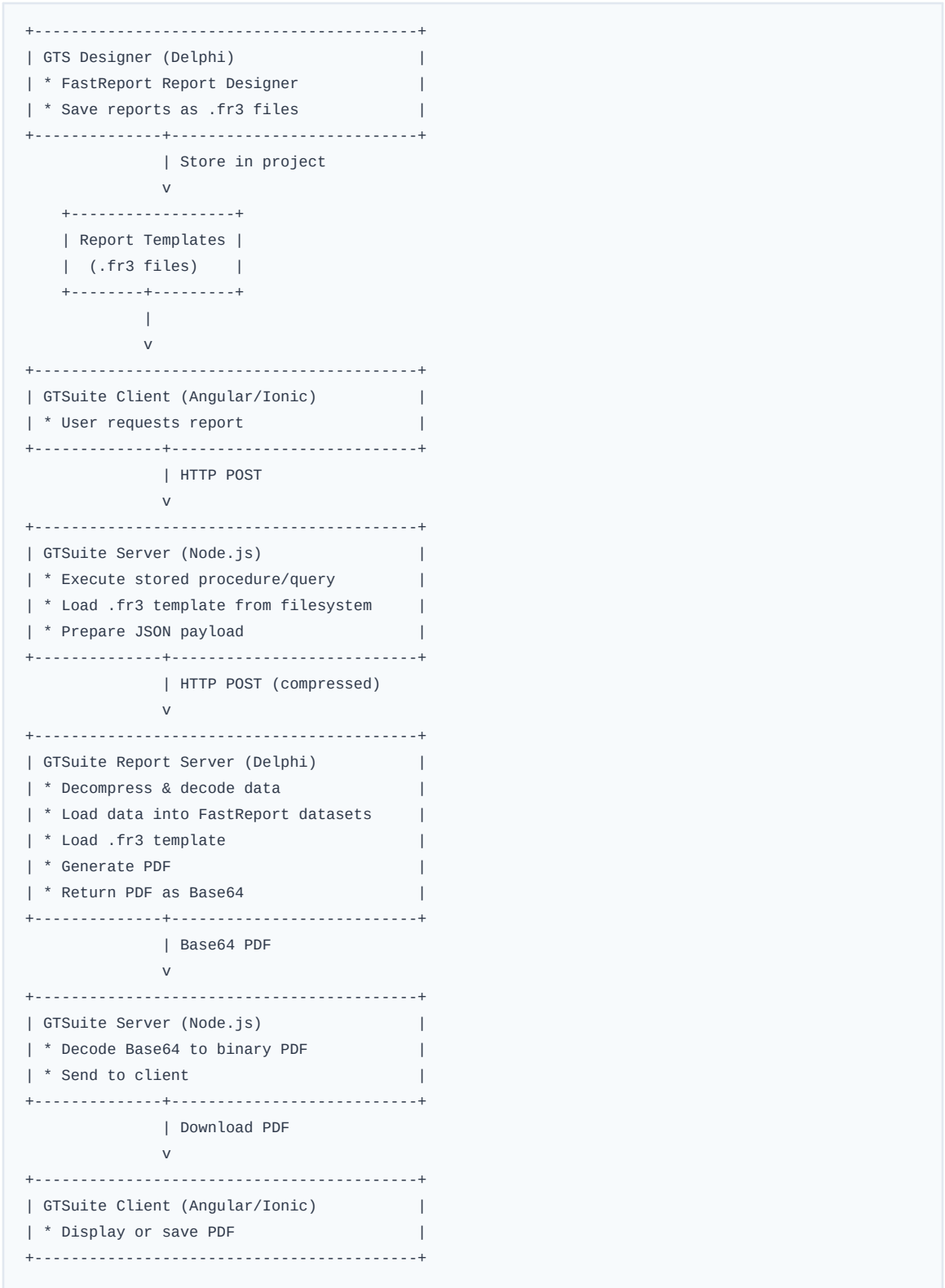
Key Characteristics

- **Standalone Service:** Independent HTTP server running on configurable port
- **FastReport VCL:** Professional report designer and PDF generator
- **Streaming Architecture:** Receives data and templates via HTTP streams
- **Multi-Dataset Support:** Handles master-detail reports with up to 10 datasets
- **Security:** IP whitelist restricts access to authorized servers only
- **High Performance:** Efficient PDF generation with compression
- **Format Support:** PDF primary, RTF optional
- **Designer Integration:** Report templates designed in GTS Designer

Architecture & Integration

Position in GTSuite Ecosystem

The Report Server acts as a specialized service layer between the Node.js backend and the client applications. It offloads the CPU-intensive PDF generation from Node.js, leveraging Delphi's native performance and FastReport's mature rendering engine.


GTS Designer (Delphi)
* FastReport Report Designer
* Save reports as .fr3 files
Store in project
Report Templates
(.fr3 files)
GTSuite Client (Angular/Ionic)
* User requests report
HTTP POST
GTSuite Server (Node.js)
* Execute stored procedure/query
* Load .fr3 template from filesystem
* Prepare JSON payload
HTTP POST (compressed)
GTSuite Report Server (Delphi)
* Decompress & decode data
* Load data into FastReport datasets
* Load .fr3 template
* Generate PDF
* Return PDF as Base64
Base64 PDF
GTSuite Server (Node.js)
* Decode Base64 to binary PDF
* Send to client
Download PDF
GTSuite Client (Angular/Ionic)
* Display or save PDF

Technology Stack

Component	Technology	Purpose
Framework	Delphi 11+	Application development platform
HTTP Server	Indy (IdHTTPWebBrokerBridge)	HTTP request handling
Report Engine	FastReport VCL 2024+	PDF generation and rendering
Data Access	FireDAC MemTable	In-memory dataset management
Web Services	DataSnap	RPC method exposure
Compression	ZLib	Data stream compression
Encoding	Base64	Binary data encoding
Export	frxPDFExport	PDF export component

FastReport VCL

FastReport is a professional report generator with a visual designer that enables creation of sophisticated reports with:

- Visual WYSIWYG report designer integrated in Delphi IDE
- Master-detail and nested subreport support
- Rich formatting: fonts, colors, borders, backgrounds
- Charts, barcodes, images, shapes
- Calculated fields and expressions
- Grouping, sorting, filtering
- Page headers, footers, summaries
- Cross-tab reports and matrix views

Application Components

Component	File	Purpose
Main Program	GTSRptServer.dpr	Console application entry point
Server Methods	ServerMethodsUnit1.pas	GetReport RPC method implementation
Web Module	WebModuleUnit1.pas	HTTP request routing and security
Report Library	GTSRptServerLib.pas	FastReport integration and PDF generation
Report Form	GTSFRGetReportForm.pas	Report form with datasets
Configuration	GTSRptServer.ini	Server settings and IP whitelist

Dataset Architecture

The Report Server supports complex master-detail reports through multiple datasets:

- **qMainSess:** Primary dataset for main report data
- **qParam:** Report parameters and configuration
- **qSubRep1-9:** Nine additional datasets for subreports and detail bands
- **frxDBDataset*:** FastReport dataset wrappers for each MemTable

Data Flow & Protocol

Request Format

The Report Server receives HTTP POST requests with JSON payload containing compressed and encoded data:

```
{
  "data": "Base64(ZLib(Base64(JSON({
    reportName: "INVOICE_01",
    reportFile: "Base64(.fr3 template)",
    procResult: {
      P_MAIN_SESS: [...],   // Main dataset rows
      P_SR01_SESS: [...],   // Subreport 1 data
      P_SR02_SESS: [...],   // Subreport 2 data
      ...
    },
    params: {
      paramName: value,
      ...
    }
  })))))"
}
```

Processing Steps

1. **Receive:** HTTP POST with Base64-encoded compressed data
2. **Decode:** Base64 decode outer layer
3. **Decompress:** ZLib decompress to get Base64 JSON
4. **Decode Again:** Base64 decode inner JSON
5. **Parse JSON:** Extract report name, template, and datasets
6. **Load Template:** Decode and load .fr3 file into FastReport
7. **Populate Datasets:** Create MemTables from JSON arrays
8. **Generate PDF:** Execute FastReport rendering
9. **Encode PDF:** Convert binary PDF to Base64
10. **Return:** JSON response with Base64 PDF

Response Format

```
{
  "valid": true,
  "data": "Base64(PDF binary content)"
}
```

Report Design Workflow

Designing Reports in GTS

Reports are designed directly within the GTS Designer application using the integrated FastReport designer:

1. **Open GTS Designer:** Launch the Delphi designer application
2. **Navigate to Reports:** Open the Report Builder module
3. **Create Report:** New report or edit existing
4. **Connect Data:** Link to report datasets (qMainSess, qSubRep1, etc.)
5. **Design Layout:** Use visual designer to place bands, fields, images
6. **Add Logic:** Calculated fields, conditional formatting, grouping
7. **Test Report:** Preview with sample data
8. **Save Template:** Save as .fr3 file in project folder
9. **Register in Metadata:** Add report definition to GtsRptReports table
10. **Upload Metadata:** Upload project to Node.js server

Report Storage

Report templates are stored as physical .fr3 files alongside the project metadata:

```
Project Structure:
/Projects/
  /GTSW/
    GTS_GTSW.db           # Metadata database
    /Reports/
      USER_LIST.fr3       # Report template
      ROLE_AUDIT.fr3
  /GTR/
    GTS_GTR.db
    /Reports/
      INVOICE_01.fr3
      INVOICE_SUMMARY.fr3
```

Configuration & Security

GTSRptServer.ini

The Report Server is configured via GTSRptServer.ini file located in the application directory:

```
[SECURITY]
PORT=8080
ALLOWED_IP_1=127.0.0.1
ALLOWED_IP_2=192.168.1.100
ALLOWED_IP_3=10.0.0.50
ALLOWED_IP_4=172.16.0.10
```

Parameter	Description	Default
PORT	HTTP listening port	8080
ALLOWED_IP_1	First allowed IP address	127.0.0.1
ALLOWED_IP_2	Second allowed IP address	127.0.0.1
ALLOWED_IP_3	Third allowed IP address	127.0.0.1
ALLOWED_IP_4	Fourth allowed IP address	127.0.0.1

Security Features

- **IP Whitelist:** Only specified IP addresses can access the service
- **Localhost Access:** Always allows 127.0.0.1 and ::1 (IPv6 localhost)
- **No Authentication:** Security relies on network isolation and IP filtering
- **Internal Network:** Should be deployed on internal network only
- **Firewall:** Recommended to use firewall rules as additional layer

Deployment & Operations

Installation

1. **Prerequisites:** Windows Server with .NET Framework
2. **FastReport License:** Ensure valid FastReport VCL license
3. **Copy Files:** Deploy GTSRptServer.exe and dependencies
4. **Configure:** Edit GTSRptServer.ini with proper settings
5. **Test Run:** Execute GTSRptServer.exe manually
6. **Verify:** Check console output for successful startup
7. **Test Endpoint:** Call GetReport method from Node.js
8. **Install Service:** Optionally install as Windows Service

Running as Console

The Report Server runs as a console application with interactive commands:

```
Available Commands:
start     - Start HTTP server
stop      - Stop HTTP server
status    - Display server status
help      - Show commands
exit      - Stop server and exit
```

Monitoring

- **Console Output:** Real-time activity displayed in console window
- **Log File:** GTSRptServer.log contains detailed operation logs
- **Performance:** Monitor CPU and memory usage during peak times
- **Errors:** Check log file for exceptions and failures
- **Health Check:** Periodic test requests from Node.js server

Node.js Server Integration

API Endpoint

The Node.js server communicates with the Report Server via DataSnap RPC:

```
POST http://localhost:8080/datasnap/rest/TServerMethods1/GetReport

Content-Type: application/json

{
  "data": "Base64Compressed JSON payload"
}
```

Node.js Implementation Pattern

Typical Node.js code to call Report Server:

```
async function generateReport(reportName, data, templatePath) {
  // 1. Read .fr3 template file
  const template = fs.readFileSync(templatePath);
  const templateB64 = template.toString('base64');

  // 2. Prepare payload
  const payload = {
    reportName: reportName,
    reportFile: templateB64,
    procResult: data.datasets,
    params: data.parameters
  };

  // 3. Compress and encode
  const json = JSON.stringify(payload);
  const b64 = Buffer.from(json).toString('base64');
  const compressed = zlib.deflateSync(b64);
  const final = compressed.toString('base64');

  // 4. Call Report Server
  const response = await axios.post(
    'http://localhost:8080/datasnap/rest/TServerMethods1/GetReport',
    { data: final }
  );

  // 5. Decode response
  const pdfBase64 = response.data.data;
  const pdfBuffer = Buffer.from(pdfBase64, 'base64');

  return pdfBuffer;
}
```

Troubleshooting

Server Won't Start

- Check if port 8080 is already in use
- Verify GTSRptServer.ini exists and is valid
- Check Windows Firewall settings
- Run as Administrator if needed

Connection Refused

- Verify server is running (check console)
- Check IP address in ALLOWED_IP_* settings
- Ensure Node.js server IP is whitelisted
- Test with curl or Postman from allowed IP

Report Generation Fails

- Check GTSRptServer.log for error details
- Verify .fr3 template file exists and is valid
- Ensure data format matches report expectations
- Test report in Designer with same data

Memory Issues

- Monitor memory usage during report generation
- Consider increasing process memory limit
- Optimize reports to use less memory
- Limit concurrent report generations

Conclusion

The GTSuite Report Server completes the GTSuite ecosystem by providing professional PDF generation capabilities through FastReport VCL. Its standalone architecture ensures: • **Performance:** Native Delphi code for fast PDF generation • **Scalability:** Can be deployed on separate server for load distribution • **Integration:** Seamless integration with GTS Designer and Node.js backend • **Professional Output:** FastReport ensures high-quality, pixel-perfect PDFs • **Flexibility:** Visual designer enables complex report layouts The Report Server demonstrates GTSuite's modular architecture where specialized components can be deployed independently based on requirements. For high-volume reporting scenarios, multiple Report Server instances can be load-balanced. Together with the Designer, Server, and Client, the Report Server completes a comprehensive platform for building enterprise applications with sophisticated reporting capabilities.

Contact & Support GTsoftware di Giancarlo Thiella For inquiries, support, or FastReport licensing